Position Paper: PIMS Migratable Threadlets

Terry Jones[1], Laxmikant Kale[2]

trj @ ornl.gov[1]

kale @ illinois.edu[2]

Problem statement:
Emerging energy concerns dictate that the cost of data movement must be carefully weighed against the benefit, and that work must proceed with minimal movement to meet power constraints. Moreover, scientific applications are frequently dependent upon regular accesses and simple neighbor-based operations (stencil operations, jacobian, finite element relaxation, …) for both adaptive mesh and fixed-mesh distributed data. Finally, technology trends including 3D memory are suggesting new ways to deal with the memory wall, but thus far an execution model is lacking.

Proposed approach:
We propose ***threadlets***, an innovative extension of ideas successfully demonstrated in the Charm++ model. A threadlet consists of a small instruction stream and its associated data. It is designed to ***enable*** Processor-in-Memory-Stack (PIMS) architectures made possible recently. Advances spurred by Moore's law make this strategy a viable alternative and multiple companies have expressed interest and/or are researching the required hardware (e.g., Micron, IBM, QlikView).

A threadlet optionally has a stack associated with it. It can be thought of as a "continuation". It consumes data and it produces/publishes data. Data produced can be tagged/marked for (a specific "port" of) another named threadlet, OR it could be just published with a global name (kind of like tuple-pace tuples, or Charisma parameters). A threadlet may be waiting for specific data items. It can either migrate to where the data items are, or fetch them, or arragnge for them to be delivered to it.

The runtime system manages movement of data as well as threads. It decides if the threadlet will move to data or the other way around. In particular, it may decided to plan a threadlet in the memory unit's control (logic) portion.

The overall flavor is that of macro-dataflow. The names of entities (data items threadlets) and their location service has to be scalably managed. This is another task for the runtime system associated with this approach.

Although chares are migratable across processors, they are mostly anchored to specific processor, and migrate occasionally. As such, they can be located using somewhat simpler mechanisms. Threadlets as conceived here are much more dynamic, requiring new sophisticated strategies to ensure threadlets and the data they need "meet" in convenient places, with minimized energy schedules.

Related Work:
In previous research, we developed Charm++ [1] and Adaptive MPI (AMPI, an implementation of the MPI standard based on Charm++) [2]. Both Charm++ and AMPI present a parallel programming system aimed at enhancing productivity in parallel programming while enhancing

scalable parallel performance. A guiding principle behind the design of Charm++ is to automate what the "system" can do best, while leaving to the programmers what they can do best. In particular, we believe that the programmer can specify ***what*** to do in parallel relatively easily, while the system can best decide which processors own which data units and which work units they execute. This approach requires an ***intelligent runtime system***, which Charm++ provides.

At its core, Charm++ employs the idea of "processor virtualization" based on migratable objects. This idea leads to programs that automatically respect locality, in part because objects provide a natural encapsulation mechanism. At the same time, it empowers the runtime system to automate resource management. The combination of features in Charm++ has made it suitable for the expression of parallelism over a range of architectures, from desktops to existing petaFLOP-scale parallel machines. Moreover, through AMPI, those advanced features become available to "legacy" MPI applications as well.

Charm++ provides an application-independent, automatic, dynamic load balancing capability. It is based on migratable objects in Charm++ and migratable threads in AMPI [3]. By migrating existing tasks among processors, the Charm++ runtime system distributes computation uniformly across all processors taking the object load into account, while minimizing the communication between them. Similarly, the object migration capability can be leveraged to provide fault tolerance. Several fault-tolerant schemes are available for Charm++ and AMPI applications [3,4]. Most of these schemes, however, work in a reactive fashion, forcing some action after a system fault is detected.

The threadlets concept shares some characteristics with the Codelets work by Guang Gao [5], and the Concurrent Collections work by Kath Knobe [6]. Both of these efforts espouse small work units for various reasons. While our system also espouses small units of work, the fundamental raison d'erte for our work is to exploit computational progress within the memory system.

In the present work, we propose to integrate smaller units of instruction streams and its associated data streams into our adaptive runtime system, and to locate this threadlets where they can efficiently be processed by emerging hardware strategy of Processing-in-Memory-Stack and/or Processing-near-Memory as a means to dramatically lower the system energy of the overall processing task while exploiting higher levels of concurrency automatically.

Assessment:
This approach directly addresses communication power concerns, readily exposes available concurrency, and through Charm++ like migratable mechanisms provides resiliency and load-balancing. It is able to benefit from prior work in the field of data flow and data flow graphs. It prepares the way for the emergence of processing in the memory stack which is at the *proposed* stage at this point (although efforts appear to be underway at several companies).

References:

[1] Laxmikant V. Kale, Eric Bohm, Celso L. Mendes, Terry Wilmarth and Gengbin Zheng. Programming Petascale Applications with Charm++ and AMPI. In Petascale Computing: Algorithms and Applications, pg 421-441, Chapman& Hall / CRC Press, ed D Bader, 2007.

[2] Gengbin Zheng, Chao Huang, and Laxmikant V. Kalé. 2006. Performance evaluation of automatic checkpoint-based fault tolerance for AMPI and Charm++. *SIGOPS Oper. Syst. Rev.* 40, 2 (April 2006), 90-99. DOI=10.1145/1131322.1131340 http://doi.acm.org/10.1145/1131322.1131340

[3] Gengbin Zheng, Orion Sky Lawlor and Laxmikant V. Kale, "Multiple Flows of Control in Migratable Parallel Programs", 2006 International Conference on Parallel Processing Workshops (ICPPW'06), Columbus, Ohio, August 2006. Publ: IEEE Computer Society, pp. 435-444.

[4] Sayantan Chakravorty, Celso L. Mendes, Laxmikant V. Kale, Terry Jones, Andrew Tauferner, Todd Inglett, Jose Moreira, HPC-Colony: Services and Interfaces for Very Large Systems, in Operating Systems Review, vol. 40, no. 2, pp. 43-49, 2006.

[5] Stéphane Zuckerman, Joshua Suetterlein, Rob Knauerhase, and Guang R. Gao. 2011. Using a "codelet" program execution model for exascale machines: position paper. In *Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era* (EXADAPT '11). ACM, New York, NY, USA, 64-69. DOI=10.1145/2000417.2000424 http://doi.acm.org/10.1145/2000417.2000424

[6] Z. Budimlic et al. Multicore implementations of the concurrent collections programming model. In CPC, 2009.